**Data Structures and Algorithms**

**Lab-07**

**Name:** Ahmad Amjad Mughal

**Reg No:** 121672

**Class:** BSCS-6C

**Task 2:**

//Ahmad Amjad Mughal

//121672

//BSCS-6C

//Task-2

#include<iostream>

#include<stackArray>

#include<string>

using namespace std;

# define length\_PostFix 50 /\* For total no of characters in string\*/

int stackArray[150];

int current = -1;

void push(char data) {

stackArray[++current] = data;

}

int pop() {

return stackArray[current--];

}

void peek() {

cout <<endl << "The toper element of stackArray is: " << stackArray[current];

}

int top() {

return stackArray[current];

}

bool isEmpty() {

return (current == -1);

}

bool isFull() {

return (current > 100);

}

// Function to convert Infix to postfix expression

string IfixTopfix(string expression);

// Function to verify that one operator is higher than the other one

int Hprecedence(char opererand1, char opererand2);

// Function to verify whether a character is operator symbol or not

bool IsOperator(char c);

// Function to verify whether a character is letter or numeric digit or not.

bool IsOperand(char c);

// Function to evaluate Postfix expression and return output

string IfixToperfix(string expression)

{

string postfix = ""; // Initialize postfix as empty string.

for (int i = 0; i< expression.length(); i++) {

// Scanning each character from left.

if (expression[i] == ' ' || expression[i] == ',') {

postfix += ' ';

continue;

}

// If character is operator, pop two elements from stackArray, perform operation and push the result back.

else if (IsOpererator(expression[i]))

{

while (!isEmpty() && top() != '(' && Hprecedence(toper(), expression[i]))

{

postfix += top();

pop();

}

push(expression[i]);

}

else if (IsOperand(expression[i])) //if character is operand then else statement executes

{

postfix += expression[i];

}

else if (expression[i] == '(')

{

push(expression[i]);

}

else if (expression[i] == ')')

{

while (!isEmpty() && top != '(') {

postfix += top();

pop();

}

postfix += ' ';

pop();

}

} //for loop lasts here

while (!isEmpty()) {

postfix += toper();

pop();

}

return postfix;

}

// Function to verify whether a character is english letter or numeric digit.

bool IsOperand(char c)

{

if (c >= '0' && c <= '9')

return true;

if (c >= 'a' && c <= 'z')

return true;

if (c >= 'A' && c <= 'Z')

return true;

return false;

}

// Function to verify whether a character is operator symbol or not.

bool IsOperator(char c)

{

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')

return true;

else

return false;

}

// Function to verify whether an operator is right associative or not.

int IsRightAssociative(char oper)

{

if (oper == '^')

return true;

return false;

}

// Function to get weight of an operator. An operator with higher weight will have higher precedence.

int GetWeight(char oper)

{

int w = -1;

switch (oper)

{

case '+':

case '-':

w = 1;

case '\*':

case '/':

w = 2;

case '^':

w = 3;

}

return w;

}

// Function to perform an operation and display output.

int Hprecedence(char operand1, char operand2)

{

int weightOperator1 = GetWeight(operand1);

int weightOperator2 = GetWeight(operand2);

// If operators have equal precedence, return true if they are left associative.

// return false, if right associative.

// operator with same precedence then left one should come first in stackArray.

if (weightOpererator1 == weightOpererator2)

{

if (IsRightAssociative(opererand1))

return false;

else

return true;

}

return weightOpererator1 > weightOpererator2 ? true : false;

}

void PostfixEvaluationFunction(string postfix)

{

int i;

//char character;

int data;

int operand1, operand2;

// evaluate postfix expression

for (i = 0; postfix[i] != '\0'; i++)

{

if (postfix[i] == ' ' || postfix[i] == ',')

continue;

else if (isdigit(postfix[i]))

{

/\*an operand,PushFunction the digit onto stackArray character - '0' is used for getting digit rather than ASCII code of digit \*/

push(postfix[i] - '0');

}

else if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '\*' || postfix[i] == '/' || postfix[i] == '^')

{

/\*we saw an operator PopFunction toper element X and next-to-toper elemnet Y from stackArray and compute Y operator X\*/

operand1 = pop();

operand2 = pop();

//character is an operator

switch (postfix[i])

{

case '^':

data = pow(operand2, operand1);

break;

case '\*':

data = operand2 \* operand1;

break;

case '/':

data = operand2 / operand1;

break;

case '+':

data = operand2 + operand1;

break;

case '-':

data = operand2 - operand1;

break;

}

//PushFunction the data obtained above onto the stackArray

push(data);

}

}

cout << "\nResult of postfix expression evaluation : " << poper();

cout << "\n\n";

}

bool IsPair(char first, char last)

{

if (first == '(' && last == ')') return true;

else if (first == '{' && last == '}') return true;

else if (first == '[' && last == ']') return true;

return false;

}

bool AreParanthesesBalanced(string exp)

{

for (int i = 0; i<exp.length(); i++)

{

if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')

push(exp[i]);

else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']')

{

if (isEmpty() || !IsPair(toper(), exp[i]))

return false;

else

poper();

}

}

return isEmpty() ? true : false;

}

int main()

{

string expression;

cout << "Enter Infix Expression: \n";

getline(cin, expression);

if (AreParanthesesBalanced(expression))

cout << "The entered expression is valid:\n";

else

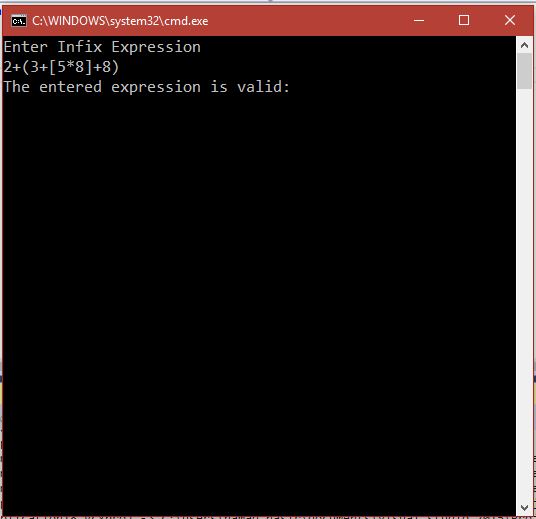
cout << "The entered expression is invalid:\n";

getchar();

return 0;

}

**Output:**

****

**Task 3:**

//Ahmad Amjad Mughal

//121672

//BSCS-6C

//Task 3

#include<iostream>

#include<stackArray>

#include<string>

using namespace std;

# define length\_PostFix 50 /\* total number of characters in input expression\*/

int stackArray[150];

int current = -1;

void push(char data) {

stackArray[++current] = data;

}

int pop() {

return stackArray[current--];

}

void peek() {

cout << endl << "The toper element of stackArray is: " << stackArray[current];

}

int top() {

return stackArray[current];

}

bool isEmpty() {

return (current == -1);

}

bool isFull() {

return (current > 100);

}

// Function to convert Infix expression to postfix

string IfixTopfix(string expression);

// Function to verify whether an operator has higher precedence over other

int Hprecedence(char operand1, char operand2);

// Function to verify whether a character is operator symbol or not.

bool IsOperator(char c);

// Function to verify whether a character is alphanumeric chanaracter (letter or numeric digit) or not.

bool IsOperand(char c);

// Function to evaluate Postfix expression and return output

string IfixTopfix(string expression)

{

string postfix = ""; // Initialize postfix as empty string.

for (int i = 0; i< expression.length(); i++) {

// Scanning each character from left.

if (expression[i] == ' ' || expression[i] == ',') {

postfix += ' ';

continue;

}

// If character is operator, pop two elements from stackArray, perform operation and push the result back.

else if (IsOperator(expression[i]))

{

while (!isEmpty() && top() != '(' && Hprecedence(top(), expression[i]))

{

postfix += top();

pop();

}

push(expression[i]);

}

// Else if character is an operand then it executes

else if (IsOperand(expression[i]))

{

postfix += expression[i];

}

else if (expression[i] == '(')

{

push(expression[i]);

}

else if (expression[i] == ')')

{

while (!isEmpty() && top() != '(') {

postfix += top();

pop();

}

postfix += ' ';

pop();

}

} //last for loop

while (!isEmpty()) {

postfix += top();

pop();

}

return postfix;

}

// Function to verify whether a character is english letter or numeric digit.

// We are assuming that opererand will be a single character

bool IsOperand(char c)

{

if (c >= '0' && c <= '9')

return true;

if (c >= 'a' && c <= 'z')

return true;

if (c >= 'A' && c <= 'Z')

return true;

return false;

}

// Function to verify whether a character is operator symbol or not.

bool IsOperator(char c)

{

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')

return true;

else

return false;

}

// Function to verify whether an operator is right associative or not.

int IsRightAssociative(char oper)

{

if (oper == '^')

return true;

return false;

}

// Function to get weight of an operator. An operator with higher weight will have higher precedence.

int GetWeight(char oper)

{

int w = -1;

switch (oper)

{

case '+':

case '-':

w = 1;

case '\*':

case '/':

w = 2;

case '^':

w = 3;

}

return w;

}

// Function to perform an operation and return output.

int Hprecedence(char operand1, char operand2)

{

int weightOperator1 = GetWeight(operand1);

int weightOperator2 = GetWeight(operand2);

// If operators have equal precedence, return true if they are left associative.

// return false, if right associative.

// if operator is left-associative, left one should be given priority.

if (weightOperator1 == weightOperator2)

{

if (IsRightAssociative(operand1))

return false;

else

return true;

}

return weightOperator1 > weightOperator2 ? true : false;

}

void PostfixEvaluationFunction(string postfix)

{

int i;

//char character;

int data;

int operand1, operand2;

// evaluate postfix expression

for (i = 0; postfix[i] != '\0'; i++)

{

//character = postfix[i];

if (postfix[i] == ' ' || postfix[i] == ',')

continue;

else if (isdigit(postfix[i]))

{

/\*an opererand,PushFunction the digit onto stackArray character - '0' is used for getting digit rather than ASCII code of digit \*/

push(postfix[i] - '0');

}

else if (postfix[i] == '+' || postfix[i] == '-' || postfix[i] == '\*' || postfix[i] == '/' || postfix[i] == '^')

{

operand1 = pop();

operand2 = pop();

//character is an operator

switch (postfix[i])

{

case '^':

data = pow(operand2, operand1);

break;

case '\*':

data = operand2 \* operand1;

break;

case '/':

data = operand2 / operand1;

break;

case '+':

data = operand2 + operand1;

break;

case '-':

data = operand2 - operand1;

break;

}

//PushFunction the data obtained above onto the stackArray

push(data);

}

}

cout << "\nResult of postfix expression evaluation : " << pop();

cout << "\n\n";

}

bool IsPair(char first, char last)

{

if (first == '(' && last == ')') return true;

else if (first == '{' && last == '}') return true;

else if (first == '[' && last == ']') return true;

return false;

}

bool AreParanthesesBalanced(string exp)

{

for (int i = 0; i<exp.length(); i++)

{

if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')

push(exp[i]);

else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']')

{

if (isEmpty() || !IsPair(top(), exp[i]))

return false;

else

pop();

}

}

return isEmpty() ? true : false;

}

int main()

{

string expression;

cout << "Enter Infix Expression: \n";

getline(cin, expression);

//if (AreParanthesesBalanced(expression))

//cout << "The entered expression is valid:\n";

//else

//cout << "The entered expression is invalid:\n";

string postfix = IfixTopfix(expression);

cout << "The postfix expression is: " << postfix << "\n";

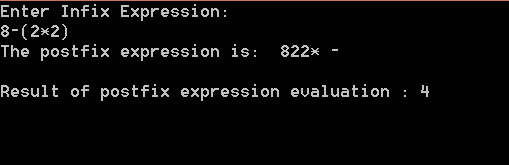
PostfixEvaluationFunction(postfix);

getchar();

return 0;

}

**Output:**

****